BACCALAUREAT

SESSION 2022

Épreuve de l'enseignement de spécialité

NUMERIQUE et SCIENCES INFORMATIQUES

Partie pratique

Classe Terminale de la voie générale

Sujet n°17

DUREE DE L'EPREUVE : 1 heure

Le sujet comporte 2 pages numérotées de 1 / 4 à 4 / 4
Dès que le sujet vous est remis, assurez-vous qu'il est complet.

Le candidat doit traiter les 2 exercices.

Exercice 1

Pour cet exercice :

- On appelle « mot » une chaîne de caractères composée avec des caractères choisis parmi les 26 lettres minuscules ou majuscules de l'alphabet,
- On appelle « phrase » une chaîne de caractères :
  - composée avec un ou plusieurs « mots » séparés entre eux par un seul caractère espace `' '`,
  - se finissant :
    - soit par un point `'.'` qui est alors collé au dernier mot,
    - soit par un point d'exclamation `'!'` ou d'interrogation `'?'` qui est alors séparé du dernier mot par un seul caractère espace `' '`.

Voici quatre exemples de phrases :

- `'Le point d exclamation est separe !'`
- `'Il y a un seul espace entre les mots !'`
- `'Le point final est colle au dernier mot.'`
- `'Gilbouze macarbi acra cor ed filbuzine ?'`

Après avoir remarqué le lien entre le nombre de mots et le nombres de caractères espace dans une phrase, programmer une fonction `nombre_de_mots` qui prend en paramètre une phrase et renvoie le nombre de mots présents dans cette phrase.

Exemples :

```
>>> nombre_de_mots('Le point d exclamation est separe !')
6

>>> nombre_de_mots('Il y a un seul espace entre les mots !')
9
```

Exercice 2

La classe `ABR` ci-dessous permet d'implémenter une structure d'arbre binaire de recherche.

```
class Noeud:
    ''' Classe implémentant un noeud d'arbre binaire
    disposant de 3 attributs :
    - valeur : la valeur de l'étiquette,
    - gauche : le sous-arbre gauche.
    - droit : le sous-arbre droit. '''

    def __init__(self, v, g, d):
        self.valeur = v
        self.gauche = g
        self.droite = d
```

```
class ABR:
    ''' Classe implémentant une structure
    d'arbre binaire de recherche. '''

    def __init__(self):
        '''Crée un arbre binaire de recherche vide'''
        self.racine = None

    def est_vide(self):
        '''Renvoie True si l'ABR est vide et False sinon.'''
        return self.racine is None

    def parcours(self, tab = []):
        ''' Renvoie la liste tab complétée avec tous les
        éléments de l'ABR triés par ordre croissant. '''

        if self.est_vide():
            return tab
        else:
            self.racine.gauche.parcours(tab)
            tab.append(...)
            ...
            return tab

    def insere(self, element):
        '''Insère un élément dans l'arbre binaire de recherche.'''
        if self.est_vide():
            self.racine = Noeud(element, ABR(), ABR())
        else:
            if element < self.racine.valeur:
                self.racine.gauche.insere(element)
            else :
                self.racine.droite.insere(element)

    def recherche(self, element):
        '''
        Renvoie True si element est présent dans l'arbre
        binaire et False sinon.
         '''
        if self.est_vide():
            return ...
        else:
            if element < self.racine.valeur:
                return ...
            elif element > self.racine.valeur:
                return ...
            else:
                return ...
```

Compléter les fonctions récursives `parcours` et `recherche` afin qu'elles respectent leurs spécifications.

Voici un exemple d'utilisation :

```
>>> a = ABR()
>>> a.insere(7)
>>> a.insere(3)
>>> a.insere(9)
>>> a.insere(1)
>>> a.insere(9)
>>> a.parcours()
[1, 3, 7, 9, 9]

>>> a.recherche(4)
False

>>> a.recherche(3)
True
```